AFRL-IF-RS-TR-2005-213
Final Technical Report
June 2005

COSENSE: COLLABORATIVE SENSEMAKING OF DISTRIBUTED DATA FOR RECOGNITION AND CONDITION MONITORING

Xerox Corporation

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K250

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-213 has been reviewed and is approved for publication

APPROVED: /s/

BRADLEY J. HARNISH
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2005	Final Jun 00 – Jan 05

4. TITLE AND SUBTITLE
COSENSE: COLLABORATIVE SENSEMAKING OF DISTRIBUTED DATA FOR RECOGNITION AND CONDITION MONITORING

5. FUNDING NUMBERS
C - F30602-00-C-0139
PE - 62301E
PR - K250
TA - 25
WU - A1

6. AUTHOR(S)
James Reich, Juan Liu,
Maurice Chu and Peter Cheung

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Xerox Corporation, Palo Alto Research Center (PARC)
3333 Coyote Hill Road
Palo Alto California 94304

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-213

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Bradley J. Harnish/IFGB/(315) 330-1884/ Bradley.Harnish@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
This report summarizes PARC's work on collaborative sensemaking under the DARPA SensIT program. The report is divided into two parts. Part I summarizes the original contract work which was centered on topics such as distributed multi-level feature analysis, distributed hypothesis management, and scaling metrics for distributed sensors. An extensive bibliography of the detailed papers resulting from this work is provided. Part II describes the work under the two contract extension where we proposed a distributed attention approach, spreading tasks through the network and filtering out the subset of useful data close to the data sources, proceeding from signals to symbols as we progress from data source to end user. Our architecture separates concerns of how and where to acquire the data from issues of processing. Data interpretation is separated into layers with clear interfaces. Testbed and simulations of the architecture and various algorithms are documented here.

14. SUBJECT TERMS
Sensor Networks, Distributed Target Tracking And Classification, Collaborative Sensemarking

15. NUMBER OF PAGES
45

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Table of Figures

Part I. Original Consense Project

Introduction

PARC dedicated a lot of effort on data collection experiments. In SITEX fild experiment in August 2000, PARC collected large volumes of acoustic data and made the data available to the SensIT community. These data later became test data sets for various teams. PARC built and maintained testbeds for data collection. For example, PARC constructed the SensIT West Coast testbed, containing about twenty Sensoria nodes, communicating with each other via wireless 802.11b. The testbed was constructed outdoors on PARC site. It had been used to support data collection, experiments, and debugging tasks by various teams, such as ISI/USC, UCLA, Fantastic Data, and PARC.

- The testbed supported data collection experiments by UCLA (Prof. Kung Yao's team) on beamforming.
- The testbed supported data collection experiments by ISI/USC (Prof. John Heidemann). PARC provided help on debugging of network diffusion routing algorithm and performance analysis.
- In PARC's target tracking experiment, the testbed was successfully used for detection and tracking of moving targets at real time with good accuracy. The testbed has been documented in [Liu-IPSN 2003].

In addition, PARC has constructed an indoor testbed comprising of over 100 Berkeley motes. The testbed has been used successfully for tracking local phenomena, as well as large scale phenomena such as the boundary of a shadow [Liu-WirelessComm 2003]. The emphasis of this research is to force large scale distribution, i.e., to disseminate complicated sensing and processing tasks to sensor nodes with limited abilities.

PARC has developed several simulators for distributed tracking and hypothesis management, including the following:

- A centralized simulator simulating tracking of single moving vehicle. The simulator is tested using data collected in the SITEX02 29'Palms field experiment. The data was collected from over 30 Sensoria nodes, placed along cross roads in the field. The collected data has been post-processed and successfully used to reconstruct the tracking of moving military vehicles. The simulator and the algorithm have been documented in our EuraSip paper [Liu-EuraSip 2003].
- A simulator simulating distributed operation of sensor nodes, capable of detecting, classification, and tracking of multiple interacting targets. This involves complex

hypothesis management regarding target identities. The simulator has been documented in [Liu-IPSN 2004].

- A simulator simulating a network of motes, each node with very limited computation and communication capability. Variations of this simulator have been used for tracking global phenomenon such as counting the number of signal sources [Fang-MobiHoc 2003], or for local phenomena such as tracking the contour (SLANT).

PARC has collaborated with UCLA on beamforming algorithms for microphone arrays, with UCLA/USC/ISI on network diffusion, with BBN and BAE in SITEX experiments, and with the general sensor network community in organizing workshops. PARC has organized two workshops on sensor networks.

- The first workshop was held on January 2001, focusing on collaborative signal processing.
- The second workshop, Information Processing in Sensor Networks (IPSN'03) was held on April 2003. It covers a broader spectrum, including collaborative information and signal processing, networking, programming, and various applications. It has attracted high-quality papers and received good attendance from academia, industry and government.

(1.0) Distributed multi-level feature analysis

In support of the thread **Distributed multi-level feature analysis**, the original SOW statement was:

1.0 Distributed multi-level feature analysis

1.1 Develop a distributed feature analysis algorithm (e.g. distributed PCA).

1.2 Develop a target localization algorithm (e.g. an incremental least-squares algorithm).

1.3 Develop a multi-sensor data fusion algorithm to combine either multiple sensor filter data streams, or multiple detection filter data streams, or both.

1.4 Test the feature analysis algorithm in the simulator using a signal data set.

1.5 Test the localization algorithm in the simulator using a point target position data set.

1.6 Test the multi-sensor fusion algorithm in the simulator using multi-channel sensor data set.

1.7 Conduct evaluations and document the scalability and performance effects of scaling of the overall feature analysis algorithm using the metrics in paragraph 3.0.

1.8 Investigate other approaches to feature analysis and multi-sensor fusion, developing tools and tests for analysis and comparison of approaches.

1.9 Demonstrate the feature analysis algorithm in the simulator using an experimental data set.

1.10 Develop a distributed target localization algorithm (e.g. an incremental least squares algorithm) that forms local estimates of target parameters within a sensor net and combines only those local estimates that can best improve the overall accuracy of localization.

PARC has developed a distributed algorithm for feature analysis. It computes principal components from stationary data samples using a stochastic gradient algorithm called Oja algorithm. The computation is incremental and any-time. More generally, PARC has developed methods for extracting information from sensory data. For example, for single target tracking, we have developed a sequential Bayesian filtering approach to track target locations [Liu-EuraSip 2003] based on multi-modal sensory data. For multi-target tracking, the complex tracking problem is separated into inter-leaved simpler problems of position tracking [Liu-IPSN 2004] and identity management [Shin-IPSN 2003]. Hypotheses regarding target identity are managed via an efficient data representation (identity belief matrix) and efficient sensor collaboration [Liu-PervComp 2003]. All the modules mentioned above have been tested and demonstrated on the PARC testbed (e.g., [Liu-IPSN03]) or in simulations (e.g., [Liu-IPSN04]).

PARC has developed an incremental least-squares algorithm for localization. The algorithm has been used for tracking of stationary target as well as moving targets. Coupled with proper error control method, it has also been used for node localization, where within a sensor network, just a few anchor nodes have known locations. The algorithm bootstraps to localize nodes with unknown locations. The method is reported in [Zhao-MobiHoc 2003]. Variations of the incremental least-squares method have been tested in simulations with good localization accuracy. The method is computationally inexpensive and fast.

PARC has developed several sensor fusion methods. The aforementioned incremental least squares method is one. A more elaborate method PARC has developed is a probabilistic sequential Bayesian filtering method for tracking moving targets. The Bayesian filtering method can combine multi-modal sensor data. For example, [Chu-IJHPCA 2001] and [Liu-EuraSip 2003] fuse range data from amplitude sensors with direction-of-arrival data from microphone array sensors.

The sensor information extraction methods and fusion methods have all been tested. For example, the incremental least squares method is tested successfully in node localization experiments. The probabilistic fusion method has been tested in simulations (see for example, [Chu-IJHPCA 2001] and [Zhao-SPM 2002]), using post-processed real data [Liu-EuraSip 2003], as well as real-time in PARC testbed [Liu-IPSN 2003].

The incremental least-squares method and the probabilistic sequential Bayesian filtering method are scalable to a large number of sensor nodes. For example, the incremental least-square method has been applied for node localization within a network consisting of over 100 nodes. We have also analyzed the scalability issues in probabilistic tracking, by comparing it with various methods, especially conventional centralized data fusion, and analyze the respective computation, communication, and power costs [Liu-EuraSip 2003].

Apart from these, PARC has explored the issue of sensor tasking and control, which is a very important problem in resource constrained sensor networks. PARC has designed various forms of IDSQ (information-driven sensor querying) criteria in which sensing, communication, computation, and energy resources are spent on the most informative sensors. Tasking has been described in our publications [Chu-IJHPCA 2001], [Zhao-SPM 2002], [Liu-EuraSip 2003]. Due to the local nature of physical phenomena, the sensor tasking strategies often take advantage of local sensing. Estimates of target locations are formed based on local sensor data only.

(2.0) Distributed hypothesis management

In support of the thread **Distributed hypothesis management**, the original SOW statement was:

2.0 Distributed hypothesis management.

- *2.1 Develop a data representation for organizing data from multiple sensor nodes into a form useful for verifying higher-level hypotheses (i.e. virtual sensors.)*
- *2.2 Develop a distributed data representation for hierarchies of hypotheses.*
- *2.3 Develop a distributed hypothesis management algorithm, consisting of hypothesis formation and validation and distributed data structures (e.g. dKDS) to maintain a set of distributed facts and hypotheses.*
- *2.4 Evaluate the scalability and performance effects of scaling of the hypothesis management algorithm using the metrics in paragraph 3.0.*
- *2.5 Test the hypothesis management algorithm in the simulator, using a simulated data set.*
- *2.6 Demonstrate and document the hypothesis management algorithm in the simulator, using an experimental data set.*
- *2.7 Explore other possibilities for distributed hypothesis management and data representations*
- *2.8 and develop tools for analysis and comparison of approaches.*
- *2.9 Initiate development of a distributed data representation (e.g. virtual sensors) for maintaining that relations between signals and hypotheses in a sensor net. Combined with the distributed localization algorithm developed in paragraph 1.10, and dKDS data structure developed in paragraph 2.3 to support the dynamic management of formation, maintenance, and validation of multiple hypotheses.*

PARC has developed several hypothesis management methods. One is relational tracking. The relation among objects are represented and stored in distributed sensor nodes. The hypotheses regarding object relations are managed dynamically [Guibas-SPM 2002].

Another important piece on hypothesis management is the tracking of multiple interacting targets. PARC has developed an efficient hypothesis management method for multi-target tracking. In particular, the method tracks ambiguity of track identity (which track corresponds to which target) when targets cross each other's path. Hypotheses are formed and represented as identity belief states; the representation is distributed over geographically spread sensors. Sensors performing classification to sort out identity ambiguity can be considered as virtual sensors. Ambiguity regarding track identity is actively adjusted when these virtual sensors gather classification evidence. PARC has designed a probabilistic method to dynamically maintain the identity consistency across sensors. Technical details regarding identity management are discussed in [Shin-IPSN 2003].

Compared to traditional hypothesis management schemes such as JPDA or MHT, our data representation and management scheme allow temporal inconsistency, and as a result, the scheme requires far less computation complexity and communication expense, and is more scalable. The detailed comparison between our hypothesis management method and more conventional data association methods are provided in [Shin-IPSN 2003].

The distributed identity management scheme has been implemented and validated in simulations. The simulation using simulated identity data is described in [Shin-IPSN03].

The identity management method mentioned above has been combined with the location tracking scheme mentioned in thread 1.0 to form a complete and coherent multi-target tracking system. Overview of the multi-target tracking information architecture is described in [Chu-InfoFusion 2003][Chu-Allerton 2003], and the data representation issues are described in [Liu-IPSN 2004]. Implementation-wise, the identity management scheme is supported by groups of collaborating sensors. The group collaboration scheme is described in the next thread. Hypotheses regarding identity and position have been formed, validated, or updated. The system has been tested successfully in simulations [Liu-IPSN 2004].

(3.0) Scaling metrics for distributed sensors

In support of the thread **Scaling metrics for distributed sensors**, the original SOW statement was:

3.0 Scaling metrics for distributed sensors. (See CDRL, A006)

3.1 Develop figures of merit for the scalability of the system (e.g., the effects of scaling on system performance or the effects of scaling on system resource usage).

3.2 Evaluate metrics for the algorithms and architecture used in the full experimental system described by the SenseIT community.

PARC has provided comparison for performance/scalability tradeoff. For example, [Liu-EuraSip 2003] compares different tracking system (IDSQ or centralized tracking) in terms of their tracking performance and scalability metrics such as communication and computation cost. For hypotheses management, PARC has provided comparison of distributed identity management and traditional MHT and JPDA methods in [Shin-IPSN 2003].

In addition, to build sensor network systems with good scalability, PARC has designed various models of collaboration. In these models, sensors with information about a certain phenomenon are organized into a collaboration group. This group-based architecture enjoys great scalability by restricting sensing, communication, and computation to only relevant sensors and freeing non-relevant sensors for resource conservation and multi-tasking. We have designed several collaboration models, such as geometrically-constrained group, N-hop neighborhood group, publish-subscribe group, and acquaintance group [Liu-PervComp 2003]. Some of these groups are static; others are dynamically evolving, capturing the spatial-temporal diversity in sensing. These collaboration models encapsulate the common data aggregation patterns in sensor network applications. From the programming point of view, the collaboration models provide useful abstraction from sensor network specifics and allow application programmers to focus on how to gather information from sensors and how the information should be processed.

The design of collaboration models is motivated by the tradeoff between system scalability and performance. The design principles have been described in [Liu-PervComp 2003]. The paper shows how a group should be formed and dynamically maintained over time, and how it can serve as building block of large scale scalable sensor network systems. The collaboration models have been successfully implemented in multi-target tracking simulations with around 100 sensor nodes. We have also mapped out detailed communication protocols to form geometrically-constrained group for target detection/track initiation and maintenance [Liu-IPSN 2003]. This has

been implemented in real-time outdoor tracking experiments on PARC service road. We have also designed a protocol to maintain the communication link between members of acquaintance groups [Fang-IPSN 2004] using a fishbone routing structure.

(4.0) Other efforts on collaborative sensing

PARC has put substantial effort into building large scale networks consisting of sensor nodes with limited computation, and communication abilities, such as the Berkeley motes. The operation of these networks is quite different from the sensor network consisting of PC-equivalent nodes. These networks force distributed processing and have much higher requirement on scalability and resource management. We have developed a programming model [Cheong-ASAC 2003] for motes to enhance component reusability and facilitate programming. We have also developed efficient data representation and processing algorithms based on simple computation implementable on the motes. This is called feather-weight sensing in [Zhao-ProcIEEE 2003]. Some applications such as shadow tracking [Liu-WirelessComm 2003] have been implemented on the mote testbed. Others such as peak counting [Fang-MobiHoc 2003] have been tested in simulations.

Other related topics that PARC has pursued include the following:

- Information-directed routing, related work is described in [Liu-ICASSP 2003b] [Liu-WSNA 2003].
- Embeded software for sensing and control, related work is described in [Zhao-ProcIEEE 2002][Liu-CSM 2003].
- Sensor network capability limits are investigated, described in [Liu-ICASSP 2003].

(5.0) References

[Liu-IPSN 2004] Juan Liu, Maurice Chu, Jie Liu, James Reich, and Feng Zhao, "Distributed State Representation for Tracking Problems in Sensor Networks," presented at *the 3rd International Symposium on Information Processing in Sensor Networks* (IPSN'04), Berkeley, CA, April, 2004.

[Guibas-Report 2004] Leonidas Guibas, "Progress in Relational Tracking and Reasoning for Sensor Networks".

[**Fang-IPSN 2004**] Qing Fang, Jie Liu, Leonidas Guibas, and Feng Zhao, "RoamHBA: Maintaining Group Connectivity in Sensor Networks," presented at *the 3rd International Symposium on Information Processing in Sensor Networks* (IPSN'04), Berkeley, CA, April, 2004.

[**Liu-TSJ 2004**] Juan Liu, Jie Liu, James Reich, Patrick Cheung, and Feng Zhao, "Distributed Group Management for Track Initiation and Maintenance in Target Localization Applications," *Telecommunication Systems Journal*, Kluwer Academic Publishers, 2004.

[Zhao-MobiHoc 2003] Feng Zhao, Juan Liu, Qingfeng Huang, and Yi Zou, "Fast and Incremental Node Localization in Ad hoc Networks", submitted to *ACM Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc) 2004*.

[**Liu-WirelessComm 2003**] Jie Liu, Patrick Cheung, Leonidas Guibas, and Feng Zhao, "Apply Geometric Duality to Energy Efficient Non-Local Phenomenon Awareness using Sensor Networks," *IEEE Wireless Communication*, special issue on "Wireless Sensor Networks: Theory and Systems," August 2004.

[**Zhao-ProcIEEE 2003**] Feng Zhao, Jie Liu, Juan Liu, Leonidas Guibas, and James Reich, "Collaborative Signal and Information Processing: An Information Directed Approach." *Proceedings of the IEEE*, pp.1199-1209, vol. 91, number 8, August 2003.

[**Liu-PervComp 2003**] Jie Liu, Maurice Chu, Juan Liu, James Reich, and Feng Zhao, "State-Centric Programming for Sensor-Actuator Network Systems." *IEEE Pervasive Computing*, October, 2003, pp.50-62.

[**Chu-InfoFusion 2003**] Maurice Chu, Sanjoy Mitter, Feng Zhao, "Distributed Multiple Target Tracking and Data Association in Ad Hoc Sensor Networks", *The 6th International Conference on Information Fusion,* Cairns, Queensland, Australia, July 8-11, 2003.

[**Chu-Allerton 2003**] Maurice Chu, Sanjoy Mitter, Feng Zhao, "An Information Architecture for Distributed Inference in Ad Hoc Sensor Networks", *The 41st Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 1-3, 2003.

[**Liu-WSNA 2003**] Juan Liu, Feng Zhao, and Dragan Petrovic, "Information-Directed Routing in Ad Hoc Sensor Networks", presented at the *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA),* San Diego, September 2003.

[**Liu-EuraSip 2003**] Juan Liu, James Reich, and Feng Zhao, "Collaborative In-Network Processing for Target Tracking." *EuraSip Journal on Applied Signal Processing,* special issue on sensor networks, vol. 23, no. 4, March 2003, pp378-391.

[**Cheong-ASAC 2003**] Elaine Cheong, Judy Liebman, Jie Liu, Feng Zhao, "TinyGALS: A Programming Model for Event-Driven Embedded Systems." *ACM Symposium on Applied Computing,* March, 2003.

[**Liu-CSM 2003**] Jie Liu and E. A. Lee, "Timed Multitasking for Real-Time Embedded Software." *IEEE Control System Magazine*, vol. 23, no. 1, special issue on "Advances in Software Enabled Control," Jan. 2003, pp. 65-75.

[**Fang-MobiHoc 2003**] Qing Fang, Feng Zhao, and Leonidas Guibas, "Lightweight Sensing and Communication Protocols for Target Enumeration and Aggregation." *ACM Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.

[**Liu-IPSN 2003**] Juan Liu, Jie Liu, James Reich, Patrick Cheung, and Feng Zhao, "Distributed Group Management for Track Initiation and Maintenance in Target Localization Applications." *Proceedings of 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, April, 2003.

[**Shin-IPSN 2003**] Jaewon Shin, Leonidas Guibas and Feng Zhao, "A Distributed Algorithm for Managing Multi-Target Identities in Wireless Ad-Hoc Sensor Networks." *Proceedings of 2nd International Workshop on Information Processing in Sensor Networks* (IPSN'03), April, 2003.

[**Liu-ICASSP 2003**] Juan Liu, Xenofon Koutsoukos, James Reich, and Feng Zhao, "Sensing Field: Coverage Characterization in Distributed Sensor Networks", *Proceedings of ICASSP*, April 2003.

[**Liu-ICASSP 2003b**] Juan Liu, Dragan Petrovic, and Feng Zhao, "Multi-Step Information-Directed Sensor Querying in Distributed Sensor Networks", *Proceedings of ICASSP*, April 2003.

[**Zhao-ProcIEEE 2002**] Feng Zhao, Chris Bailey-Kellogg, Markus Fromherz, "Physics-based Encapsulation in Embedded Software for Distributed Sensing and Control Applications." *Proceedings of the IEEE*, 91(1):40-63, Jan. 2002.

[**Kumar-SPM 2002**] Sri Kumar, David Shepherd, and Feng Zhao, "Collaborative Signal and Information Processing in Micro-Sensor Networks." *IEEE Signal Processing Magazine*, March 2002.

[**Zhao-SPM 2002**] Feng Zhao, Jaewon Shin, and James Reich, "Information-Driven Dynamic Sensor Collaboration for Tracking Applications." *IEEE Signal Processing Magazine*, 19(2):61-72, March 2002.

[**Guibas-SPM 2002**] Leonidas Guibas, "Sensing, tracking, and reasoning with relations." *IEEE Signal Processing Magazine,* March 2002.

[**Liu-WSNA 2002**] Jie Liu, Patrick Cheung, Leonidas Guibas, and Feng Zhao, "A Dual-Space Approach to Tracking and Sensor Management in Wireless Sensor Networks." *ACM*

International Workshop on Wireless Sensor Networks and Applications, Atlanta, September 2002. Also, Palo Alto Research Center Technical Report P2002-10077, March 2002.

[**Chu-IJHPCA 2001**] Maurice Chu, Horst Haussecker, and Feng Zhao, "Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks." *Int'l J. High Performance Computing Applications*, 16(3):90-110, Fall 2002. Also, Xerox Palo Alto Research Center Technical Report P2001-10113, May 2001.

Part II. Seedling Extensions

Introduction

This part of the final report discusses the original SOW items of the first and second extensions. Due to the overlap of many of the SOW items in the two extensions, primarily due to the fact that the SOW items in the second extension were intended to be extensions of the items in the first SOW, we will report on the related SOW items in conjunction. After each SOW item (shown in italics), we print a letter (in bold face) indicating the section which describes the completion of the SOW item.

As reported in the SOW of the original extension, we have the following items.

1. *Develop an architecture for task-driven information collection and algorithms for incremental information update and optimal allocation of sensing and communication resources.* **(See Section A)**

2. *Develop a model of normalcy and threats and associated algorithms for detecting and classifying threats.* **(See Section B)**

3. *Develop a set of metrics for evaluating the performance of the distributed attention architecture. Some of the metrics will be task specific, such as false alarm/miss rate and response time. Detection and tracking capacity of such networks will be characterized as a function of sensor density, number of targets, task priorities, sensor and network failure probabilities.* **(See Section F)**

4. *Develop a simulator to support the development of the architecture and algorithms, and validate them for a multi-target sensing problem.* **(See Section E)**

5. *Build a 20-node testbed of a sensor network, to evaluate different sensing modalities such as video and acoustic and their suitability for a larger testbed for the DARPA program at a later stage. The testbed will be based on off-the-shelf programmable PC104-class processors (e.g., Openbrick node) with USB based camera/sensor ports and linux OS for ease of programming and distribution to the larger research and development community.* **(See Section E)**

6. *Perform data collection experiments on the testbed hardware and use the data to validate the distributed attention architecture. The testbed will also be used to characterize uncertainties in sensing and threat models and their effects on the sensor network capacity and performances.* **(See Section G)**

7. *Deliver a final report to DARPA on the distributed attention architecture.* **(See Section I)**

As reported in the SOW of the second extension, we have the following items.

1. *Define an overall system architecture, comprising sensing, inference, learning, self-awareness, resource allocation and communication in a distributed setting.* **(See Section A)**

2. *Add a Cognitive Layer to the layered inference architecture, reasoning about higher-level abstract models of the behaviors that were identified by the pre-attentive layer and tracked by*

the attentive layer. This includes a capability for the cognitive level to steer the sensing of the attentive and/or pre-attentive layer. We will compare this architecture to previous work in the field. **(See Section C)**

3. *Build a framework for network self-awareness, comprising a more detailed task resource use model and a subsystem allowing nodes to detect and adjust to the sensing capabilities, processing power and/or network connectivity of their local neighborhood. Modify the resource allocation algorithms to support dynamic adaptation to these parameters.* **(See Section D)**

4. *Enhance the testbed to display the vehicle behaviors needed for cognitive processing and to support time-varying sensing capabilities and/or network topology as needed to demonstrate self-awareness.* **(See Section E)**

5. *Produce a document extending the benchmark scenarios and metrics to support the new capabilities described above.* **(See Section F)**

6. *Host a workshop at PARC to solicit input on the distributed attention architecture and generate interest in the underlying research ideas. Produce a whitepaper exploring the research issues for a DARPA program in this space.* **(See Section H)**

7. *Deliver a final report on the distributed attention architecture to DARPA.* **(See Section I)**

(6.0) Overall system architecture

The overall system architecture implemented in the testbed developed by PARC is described in Appendix I, which describes the overall architecture, design decisions, and distributed implementation of the whole system. An initial design of the architecture was also described in [Chu-IDSS2004] which we refer to serve as a summary along with references to related work. What follows is a summary of the generalization of the actual system architecture which is described in greater detail in Report A011.

From a holistic view, the distributed attention architecture makes a separation between the application side, which is responsible for the application-specific inference, sensemaking, and learning capabilities, from the system side, which is responsible for the physical sensing, networking, and resource allocation capabilities, as shown in the Figure 1.

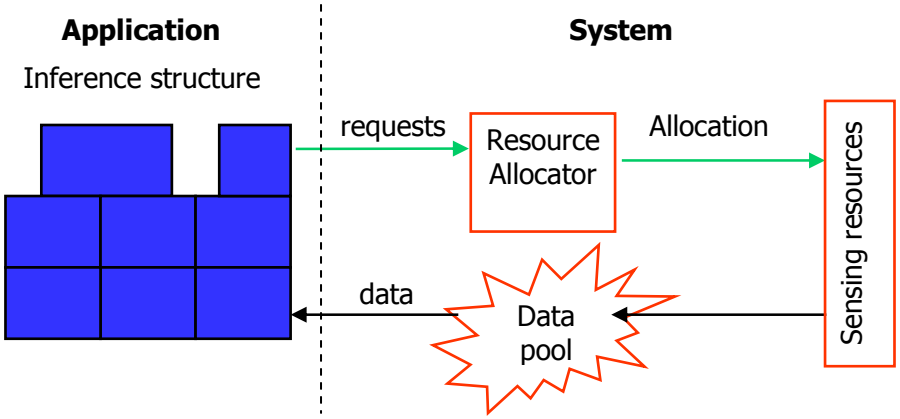


Figure 1. System architecture

The interface between these two parts is a language which can summarize the necessary parameters and requirements needed for the two sides to work together as a whole. From the application side to the system side, the application can specify requests for certain kinds of data which would be useful to its sensemaking tasks. By prioritizing these requests, the resource allocator on the system side can make an intelligent task-to-resource allocation decision, and schedule the appropriate sensors to provide the desired data. From the system side to the application side, the system can provide data tagged with attributes (like the time, location, and type of data) so that the sensemaking tasks have enough information to begin to make sense of the raw data. Thus, from the application side's point of view, the system is a data producing black box which takes requests for data as input and outputs data to the application side.

The application side maintains an inference structure, which we call a layered inference architecture. The layered inference architecture is structured so that the sensemaking algorithms are broken up into modules with the dependencies and interaction between the modules explicitly defined. In fact, a generic form of an inference module is proposed with parameters to adjust the behavior of the module and other modules connected to it. Such a decomposition is useful for two reasons. First, since the sensor network is a distributed system, the inference modules can be distributed throughout the network. The particular choice of distribution of these inference modules for processing on sensor nodes can be made in real-time so that processing and communication can be optimized to the particular sensor network resources available. This flexibility enables network self awareness since the particular choice of distributing inference modules can be adjusted when nodes and links fail. Secondly, we can change the behavior of how the sensemaking algorithms behave on-line by tweaking the parameters of the modules. The particular choice of tweaking parameters affects what sensemaking algorithms will be enabled, and hence, what sensing resources are needed. Thus, these adjustments to the parameters are how we focus attention only on objects of interest. Learning algorithms can be incorporated to change the behavior of the algorithms and how they are distributed on the sensor network, which is a form of self awareness from the sensemaking perspective.

The system side maintains knowledge of the network, the sensing resources available, and connectivity. This information can be used to maintain a network self awareness so that as new nodes enter or old nodes depart from the network, resource allocation can be adjusted to make use of only the available resources in real time. Furthermore, routing to disseminate data to the appropriate nodes, as dictated by the placement of inference modules, can be adjusted as the need arises.

(7.0) Normalcy

PARC has developed models of normalcy which are relevant at different layers of the inference stack in the distributed attention architecture. [Normalcy-2004] is a slide deck which outlines what the relevance of normalcy is on distributed attention, the approaches we consider to define normalcy, how normalcy is incorporated into the inference architecture, and some scenarios of how normalcy and inference will work together to focus attention on a subset of abnormal behaviors.

(8.0) Cognitive layer of processing

PARC has worked on the problem of detecting the source of disturbance among a group of interacting objects, which is an example of a higher-level behavior to be added to the cognitive layer of the layered inference architecture. In particular, for scenarios where there are lots of independently moving objects like people in busy city streets or vehicles on highways, PARC has considered the problem of identifying "bully" objects based on the motion of several interacting objects. The assumption is that a bully object moves along its desired path with complete disregard of other objects near it even if a collision is imminent, whereas non-bully objects are willing to stray away from their desired path in order to avoid collisions. The result is that bully objects tend to push non-bully objects out of the way causing them to stray from their desired path.

Such a source of disturbance has ramifications to the pre-attentive and attentive layers of processing. The pre-attentive layer is the first line of filtering for abnormal events in this distributed system. In the earlier distributed attention system without the cognitive layer, the pre-attentive layer tagged objects with statistically rare velocities as abnormal, based on the history of position/velocity detections of objects. The attentive layer then tracks these pre-attentively abnormal objects. In order to incorporate bully detection at the cognitive layer of processing, the cognitive layer must adjust the action of the pre-attentive and attentive layers of processing. First, a bully may not act abnormal in the "pre-attentive" sense because it could have normal position and velocity detections over time. On the other hand, the non-bullies are pushed out of the way of the bully so that they may exhibit abnormal velocities in the "pre-attentive" sense. Hence, in order to minimize the amount of unnecessary computation, once a bully is identified, the cognitive layer should alter the notion of what it means to be abnormal in the "pre-attentive" sense for objects near the bully. Second, since the pre-attentive layer may not tag the actual bully as being abnormal in the "pre-attentive" sense, there must be a way to instantiate tracks so that a bully can be tracked. Since an object which is pushed out of the way by a bully will most likely exhibit abnormality in the "pre-attentive" sense and, hence, a track instantiated for that object, the idea is for the cognitive layer to detect when a track looks like it is being pushed. Then, all other objects in the vicinity are hypothesized to be possible bullies, thus instantiating tracks for possible bullies. At this point, the cognitive layer can reason about the behavior of these tracks and declare them bullies or non-bullies.

Layered inference architectures, which work with multiple representations of data that are processed interactively, have long been proposed within the computer vision community, although little work has been explicitly published in the literature. The notions of top-down and bottom-up processing have long been considered complementary processes which if, working

together, have been hypothesized to help manage the complexity of processing in recognition problems. In particular, [Ullman-1997] suggested a machinery called counterstreams which combined top-down and bottom-up processes to work in harmony. Hierarchical control [Vachtsevanos-1998] has been used in practice which has promoted modularized design of complex control systems. Component-based software architectures [CompSoft-1998], [Szperski-1998] could enable extensible and flexible designs for future evolution. Our inference stack part of the distributed attention architecture uses these ideas to build the representations and algorithms in the pre-attentive, attentive, and cognitive layers. The primary difference in our inference stack lies in the fact that our system is distributed so that the representations and processes within each layer must themselves be modularized to promote distribution of the representations onto a sensor network. Hence, the generic modules to be composed together in the inference stack, as described in Report A011 for the purpose of maintaining a distributed representation on a distributed system are novel.

The resource allocation part can be related to the scheduling and allocation tasks of any modern operating system on computers. The main difference again comes from the fact that we are concerned with a distributed system so that scheduling and resource allocation must have a distributed implementation. Furthermore, the priorities of allocating certain tasks changes with the state of phenomenon in the environment rather than the usual fixed priorities of tasks in a traditional operating system. This dynamism is a particular feature of sensemaking algorithms on sensor networks since the tasks that are executed depends directly on the occurrence of phenomena in the environment.

(9.0) Framework for network self-awareness

Our framework for incorporating network self-awareness has manifested itself in a software architecture called the Janus architecture. See the slide deck [Janus-2004] for more details on the conceptual view of the architecture and the underlying services. The Janus architecture provides an abstraction of the sensor network so that application code, in the form of software agents, implementing inference algorithms need only register for data by specifying the desired modality of the data regardless of the type of sensor hardware or whether the sensor is on a remote node or the local node. By creating this separation between sensing resources providing data from application software agents consuming data through the indirection of a data description language, the functionality needed for network self-awareness can be separated from the code in the application agents.

The Janus services include several modules which are responsible for different functions. The two modules which support network self-awareness are the “Neighborhood” and the “Negotiator” modules. The “Neighborhood” is responsible for keeping the local database up to date of all hosts within a local neighborhood. All algorithms for detecting when hosts die or when new hosts enter the network are incorporated into this module, and this information about the local neighborhood of hosts is stored in the local database. Thus, the “Neighborhood” module is the subsystem which allows nodes to detect and maintain awareness about the dynamic connectivity and availability of nodes within a local neighborhood. The second aspect of network self-awareness is the ability for the application agents to adapt to these dynamic network conditions. The “Negotiator” module serves as the intermediary between the application agent algorithms and the dynamic network resources so that algorithms for allocating sensing resources to sensing tasks reside here. The effect of this intermediary is that the application agent code need only request for data by specifying the type and attributes of the data without knowledge of the

specific sensors in the network. The "Negotiator" can then look in the local database to see what sensors are available to provide the requested data and schedule the sensor to provide the requested data. Since the "Neighborhood" maintains information about the local neighborhood of hosts, the "Negotiator" is in real-time allocating resources for tasks according to the current state of the network. Furthermore, this separation of functionality into the "Neighborhood" and the "Negotiator" allows network self-awareness to be implemented in a modular fashion.

(10.0) Testbed and Simulator

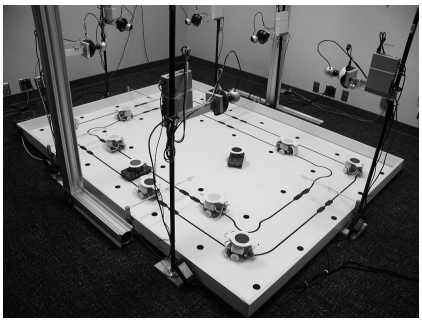


Figure 2. First generation testbed

PARC's first generation testbed is shown in Figure 2.

It served to demonstrate an exploratory prototype of a two-layered inference architecture with resource allocation. Two types of vehicles are present: distractors, represented by line-tracking robots which automatically follow tracks drawn on the testbed surface, and vehicles of interest, represented by radio controlled tanks. To simplify image processing, vehicles were marked with either a disk of a single color (giving 4 appearance models) or a disk surrounded by a ring of a different color (16 appearance models). No *a priori* knowledge of the colors assigned to distractors or vehicles of interest is used by the system.

Above and around the testbed, six sensor nodes were installed. The sensor nodes are OpenBricks, 300 MHz x86-class tiny PCs with TrackerPod pan-tilt cameras (PTCs) mounted on each. At each timestep, each PTC points at the center of one of 4 neighboring regions on the testbed, with 0-2 regions overlapping among each pair of neighboring nodes.

The size of the testbed was determined during the summer of 2003, when a smaller group of three trackerpod cameras were mounted on vertical posts on the testbed to track several battery-operated line-tracker vehicles. There were a few contradicting requirements we wanted the testbed to meet: First, it should be as large as possible to accommodate as many cameras as possible in future expansions. Second, practically it should fit inside the room which we had previously used as the anechoic chamber for the SensIT acoustic tracking project, and should leave adequate walking space around the testbed. Third, the testbed should be easily managed by one person, and occasionally by two.

To provide the option of erecting cameras in the middle of the testbed, the testbed was designed to allow cables to run underneath. Access to the space underneath the testbed was necessary, so fibre pegged boards were chosen. In addition, spacers made from 0.75 inch high medium density compressed particle boards were attached from underneath and secured by wood screws driven through the pegged holes. The spacers were generally spaced 1 foot apart and that was sufficient to support the weight of a person walking on top of the board. Nevertheless we eventually learned

that mounting a camera base in the middle of the testbed still involved too much effort of opening a big hole and fitting a relatively large spacer board to stabilize the camera as it was motorized to move. Besides, it would be inflexible to move the camera base somewhere else should the need arise. Hence we finally designed a racetrack vehicle path and opted to mount the camera bases around the perimeter and cantilevered the camera a foot or so into the testbed.

The final size of the testbed was also influenced by the material sizes that we could acquire. The pegged boards were bought in standardized size of 8 feet X 4 feet, with holes 0.25 inch in diameter and spaced 1 inch apart each other in a rectangular grid. We finally decided that the testbed be made of two standardized boards, each cut to 6 feet 1.5 inches X 4 feet and combined on the long side. Four 4-inch high boards were screwed around the perimeter as a frame and that took in 0.75 inch from each edge, hence the final working area of the testbed was measured 94.5 inches X 6 feet.

Each of the trackerpods from Eagletron Trackercam had a Logitech Quickcam Pro 4000 webcam mounted on it. The robotic tripod could pan 160 degrees and tilt 110 degrees at a speed of up to 100 degrees per second. Pan action was typically used on our testbed experiments. Both the tripod and the webcam were connected to the Openbrick by USB-1.1, hence both USB ports on an Openbrick were occupied. Additionally, the trackerpod motors drew up to 0.5 Amps from the port so it could post overload problem to some standard USB ports but the ports on the Openbrick were able to supply the necessary current. Each of the six webcams was 41 inches above the surface of the testbed. At that distance, a camera snap-shot would cover 24 inches X 18 inches if aimed vertically below camera. The robots and remote-controlled vehicles all had a printed color blob of 3-inch diameter on top to provide visual identifications. The blob was placed 4 inches from the testbed surface thus the cameras were focused to a distance of 37 inches.

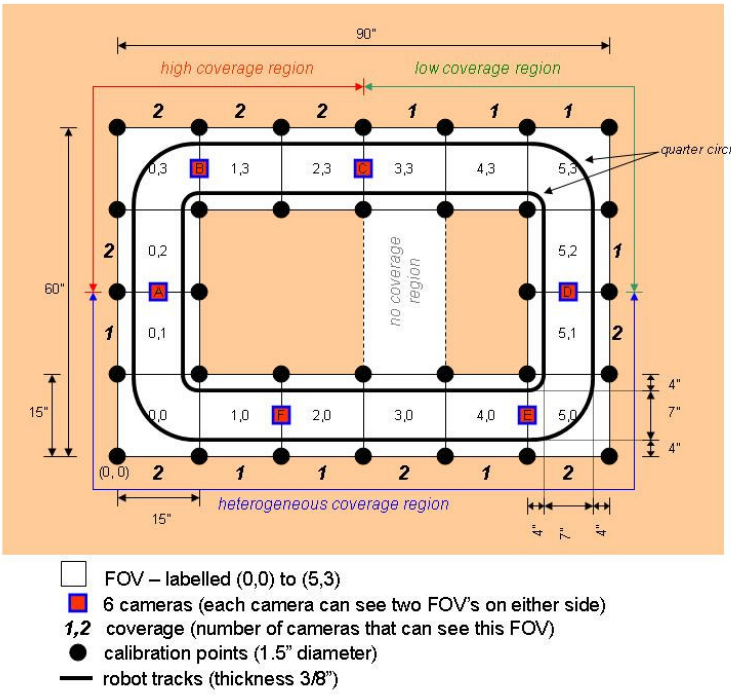


Figure 3. Testbed layout

The 32 calibration dots were also elevated 4 inches with metal stands so that all the color information were on the same plane and any image skew caused by height differences avoided.

Figure 3 above shows the layout of the testbed track. The two robot tracks were printed on large format non-reflective (180 gram) paper. We divided the 90"x60" region into 15" squares. Each square represented a single field of view (FOV) that any camera was allowed to look, and they are numbered from (0,0) representing the bottom left FOV to (5,3) representing the top right FOV. Within the testbed we wanted to accommodate as many FOVs as possible but the minimum size of a FOV was dictated by the sizes of the vehicles. Two robots or remote-controlled vehicles must be able to past each other within a FOV. Besides, we had to mark each of the four corners of a FOV with a color blob, which would have a minimum surface area to be recognized by the cameras above. Fifteen-inch square was the minimum size of a FOV that we could design for. At this dimension we were able to carve out 16 FOVs. When these 16 FOVs were to be monitored by six cameras each able to pan or tilt to four positions, 8 of the FOVs could be monitored by two cameras. A physical configuration was thus constructed for the camera network to allocate sensing resources to optimize what the entire network can see.

Robots and remote-controlled vehicles were only allowed to be in the white areas of the layout. Six cameras were positioned at each of the red squares marked with a letter, and at any given moment each camera was allowed to look in one of 4 FOV's, two in the counter-clockwise direction and two in the clockwise direction. For example, camera E can observe FOV's (3,0), (4,0), (5,0), and (5,1). Note that FOV (3,1) and (3,2) are areas where only remote-controlled vehicles were allowed to enter, and no camera is allowed to observe these no coverage FOV's.

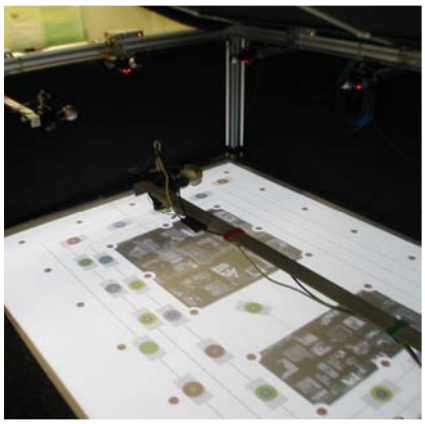


Figure 4. Second generation testbed

PARC's second generation testbed is shown in Figure 4, which also consisted of a set of networked sensor nodes mounted above the testbed, each controlling a pan-tilt camera (PTC). These nodes look directly down on a screen on which simulated images of moving vehicles are projected by a conference room projector. This combination of simulated data and a real sensor network lets us incorporate the difficult-to-simulate effects of video sensing, networking and limited computation, while permitting us to generate repeatable scenarios in which very large numbers of moving targets and distractors may behave in complex, purposeful ways, which was too difficult and time consuming to implement on the first testbed. The layout of the second generation testbed is similar to the layout of the first generation testbed. An example simulated scenario is shown below. Both distractors and targets are shown as a rectangle with a pair of coaxial, uniquely-colored dots, with no *a priori* knowledge of which colors correspond to targets and distractors provided. Regions with buildings are impassable, while the central "crossover" path is inaccessible to any of the cameras. The small red dots are solely for pan tilt camera (PTC) calibration.

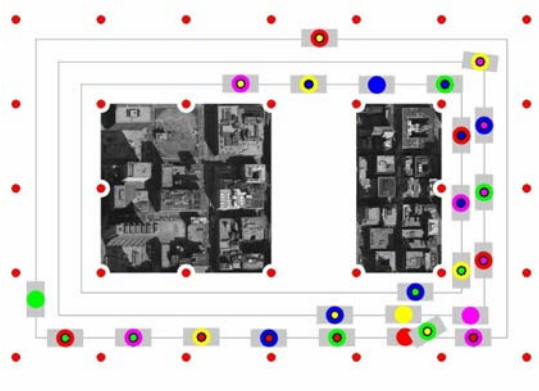


Figure 5. Example simulated scenario

The sensor nodes, as before, are OpenBricks, 300 MHz x86-class (AMD Geode) tiny PCs with TrackerPod PTCs mounted on each. These nodes currently use wired power and networking for simplicity of experimentation. Since the plane that the color blobs appear is right on the screen, the cameras are mounted only 37 inches above, and will capture the same amount of information as they did in our first generation testbed. At each timestep, each PTC points at the center of one of 4 neighboring regions (Fields of View, or FOVs) on the testbed, with 0-2 FOVs overlapping between each pair of neighboring nodes. Two frames are captured at each iteration (every 2 seconds), separated by 75 msec. The YUV4:2:0-formatted images are separated into five color planes in {Red, Green, Blue, Yellow, Magenta} using a radial basis function classifier in UV space (i.e. using only chrominance) based on five calibrated prototype colors. A simple connected-component blob detector finds the centroids of blobs in the image, and velocity estimates are made by back-differencing centroid positions of nearest neighbor blobs of the same color between the two frames.

PARC has developed a simulator for the motion of many interacting objects according to a electrical particle model of interaction modified with a goal directed motion. [Chabbra-2004] describes the simulator in more detail. This simulator generated movies of the scenarios used to demonstrate the distributed attention system.

(11.0) Benchmark scenarios and metrics

The slide deck [Scenarios-2004] shows two benchmark scenarios for our distributed attention system. The scenarios represent cases where our system is to detect a bully which has caused a commotion amongst a group of normally moving objects.

Appendix II describes performance metrics which are applicable to measuring the performance of the distributed attention system. They are primarily concerned with the performance of the inference as traded off with resource limitations and usage.

(12.0) Data collection and validation

PARC has collected data on the distributed attention system from the second generation testbed to validate that the resource allocation mechanisms do produce better performance in certain

regimes of sensor density and sensing coverage. See Appendix III for details on some experiments showing the performance of the distributed attention system.

(13.0) Whitepaper

A whitepaper has been produced which outlines a possible research thrust for DARPA. See Report A011.

(14.0) Final report

This concludes the final report on the seedling efforts.

(15.0) References

[CompSoft-1998] Component-based Software, IEEE Software special issue, volume 15, number 5, Sept-Oct. 1998.

[Shanmugan-1988] K. Sam Shanmugan and A.M. Breiphol, Random Signals: Detection, Estimation, and Data Analysis, John Wiley & Sons, Inc., 1988.

[Szperski-1998] C. Szyperski. Component Software. Addison-Wesley, 1998.

[Ullman-1997] S. Ullman, High-level Vision, Object Recognition and Visual Cognition. A Bradford Book, The MIT Press, New York, 1997.

[Vachtsevanos-1998] G. Vachtsevanos, "Hierarchical Control," Handbook of Fuzzy Computation, Eds. in Chief E.Ruspini, P. Bonissone and W. Pedrycz, Institute of Physics Publishing, pp. F 2.2:42-53, 1998.

[CHU-IDSS2004] M. Chu, J. Reich, F. Zhao, "Distributed Attention in Large Sensor Networks," Intelligent Distributed Surveillance Systems IDSS 2004, London, 2004

[Normalcy-2004] PARC presentation material on Project CD

[Janus-2004] PARC presentation material on Project CD

[Scenarios-2004] PARC presentation material on Project CD

Appendix I. Implemented system architecture for the distributed attention testbed

This appendix describes the system architecture for the current distributed attention testbed. The goal of a system architecture is to divide the different functionalities of the system into as independent, modular pieces as possible. Hence, the design considerations are based on defining boundaries that split functionality into nearly independent pieces via a well-defined interface between modules. A good system architecture promotes easier future evolution since each module can be independently designed as long as the interfaces are respected.

The sensemaking tasks currently implemented on the testbed are the following.

1. Scan the world so as to get data about abnormally behaving objects as quickly as possible.
2. Detect abnormally behaving objects.
3. Track abnormally behaving objects.

The main system characteristics are the following.

1. limited sensing coverage - The sensors in the system cannot observe all phenomena in the world at any one time. In particular, the six cameras of the testbed cannot take an image of the entire testbed.
2. distributed system – The algorithms for running the tasks must be distributed implementations. Hence, the art of designing the architecture is to modularize the computations involved so as to keep each computational component as independent from each other as possible. However, since dependencies among components are inevitable (i.e., the resulting blobs from the blob detection computations are needed by the tracking computations), it is desirable to keep the components as loosely coupled as possible (e.g., the tracking computations don't care exactly which blob detector the blob came from as long as *some* blob detector provides the data that the tracking computations require.)

1. System Architecture -- Holistic View to Distributed Implementation

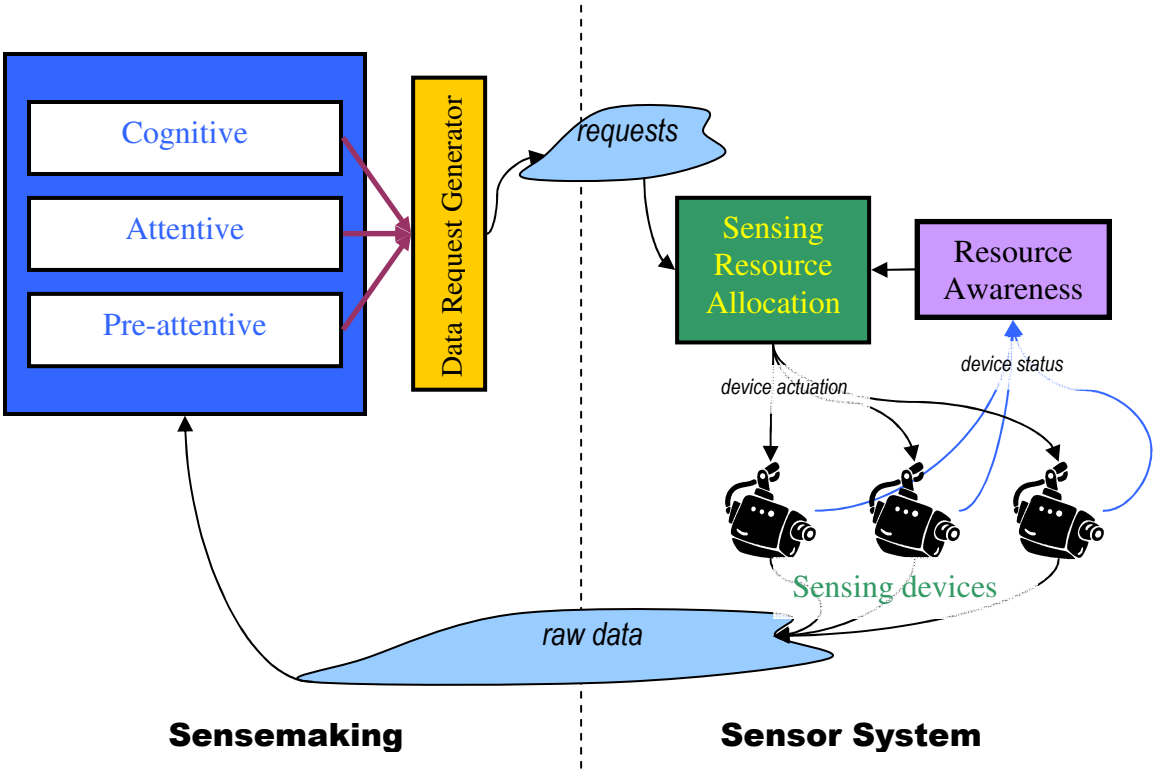


Figure 6. Overall system architecture

To accomplish the tasks above with consideration of the main system characteristics described and to allow future expansion, we have one major division of the system architecture: the sensemaking part and the sensor system part.

1.1 Sensemaking Part

The sensemaking part is the analytical part of the system which is responsible for two major functionalities:

1. interpret sensor data to form the appropriate higher-level representations reflecting the states of phenomena in the world and
2. determine what sensor data would be most useful for further interpretation.

The first functionality of the sensemaking part is embodied within the layered inference architecture in blue in Figure 6. The problem of sensor data interpretation is the subject of traditional information processing, which includes detection, estimation, and tracking. These information processing algorithms extract the desired information from the possibly noisy data in some optimal manner *without the ability to control what data is provided*. Our system, on the other hand, can potentially control what data is received so that the second functionality in the sensemaking part is to suggest what data to actively seek, which is embodied in the data request generator denoted in gold in Figure 6. This second functionality addresses the "limited sensing coverage" aspect of our system by allowing the current inferred state of the world to influence what data to collect next. Note that this functionality implements a part of the focusing of attention aspect of our distributed attention system.

Such active sensing systems are not novel and have been reported in the literature; however, one additional aspect of our system is that it is a distributed system. The design of the computations involved for the sensemaking tasks must be organized in a way which can tolerate the communication latency and unreliability of cross-node communications since sensemaking tasks may require real-time performance which can push the available communication bandwidth to the limit. For example, limited communication bandwidth may prohibit the transfer of streaming video to remote nodes so that the information in the video stream must be summarized in order to be sent. A closer look at the layered inference architecture in Figure 6 shows our three layered architecture for interpreting data. At the bottom is the pre-attentive layer; the next is the attentive layer, and the highest layer is the cognitive layer. The organizational principle for the three layers of processing involves two considerations: the semantics of the representations and the cross-node communication requirements of the computations.

Holistic View of the Pre-attentive Layer

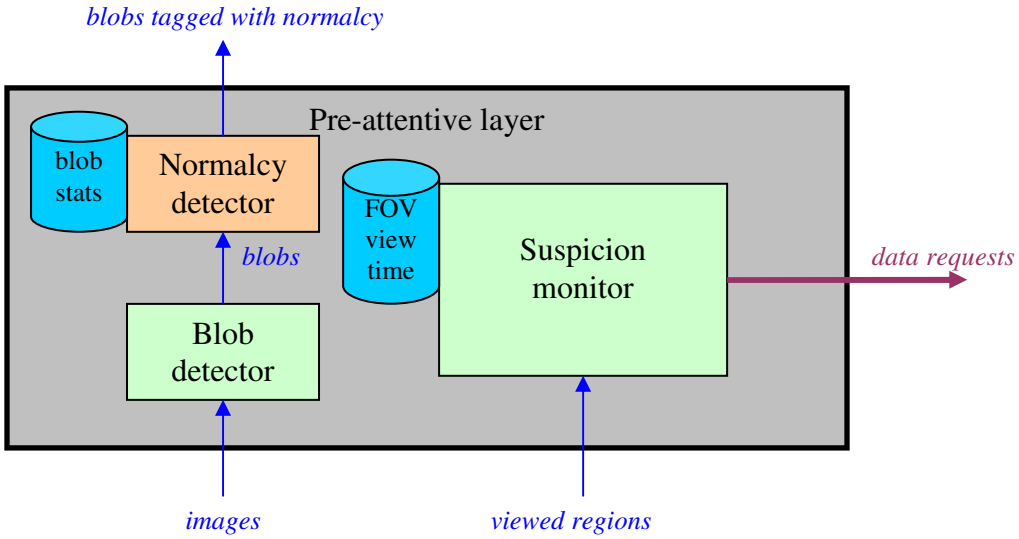


Figure 7. Pre-attentive layer

The computations in the pre-attentive layer result in representations which are semantically strongly related to the raw data streams collected by the sensor hardware. In our camera testbed, the pre-attentive layer, shown in Figure 7, takes images captured from a camera and processes them through a blob detector which computes the location, velocity, and color of colored dots in a sequence of two images. These blobs are then passed through a normalcy detector which tags a normalcy flag depending on whether the blob's location/velocity pair is considered statistically rare, based on a histogram of the history of blob location/velocity detections stored in a blob statistics database. In the current system, a separate learning phase for collecting blob detections was executed to compute the histogram stored in the blob statistics database; we imagine future extensions which allow this learning phase to occur online. Furthermore, there is a suspicion monitor which monitors which FOV's are probable for new abnormal behavior to emerge in the world. Currently, the suspicion monitor maintains the amount of time that has elapsed since the last observation of each FOV in the testbed, where longer elapsed time corresponds to higher probability of a new abnormal behavior emerging. The data for suspicion monitoring is the time that a particular FOV has been viewed by a pan-tilt camera.

Processing in the pre-attentive layer should require very little or no cross node communications. Thus, the pre-attentive layer can include processing of high data rate streams that require no cross node data. For example, blob detection requires images only from the local camera sensor so no cross node communications are required. Furthermore, any cross node communications should be relatively static in the sense that the data sources and sinks do not change rapidly. For example, the suspicion monitor of a particular FOV requires information from the set of cameras that can view the FOV, which we assume will not change rapidly. The ramifications of having none or only a few, static cross node communications is that the overhead needed to maintain the sources and sinks of data requires only an initial setup (footnote: This is the overhead from an end-to-end perspective and is true only in sensor networks with relatively stable links and nodes. However, even for cases where failures in links and nodes are frequent, static data sources and sinks are easier to maintain than dynamic ones.). Thus, the processes in the pre-attentive layer share a common characteristic in regards to their distributed implementation. Those processes requiring no cross node communications require only a replication of the computations onto each node, and those that do require data from remote nodes require only an initial set up of the data sources and sinks and an implementation to forward data via cross node communications as appropriate.

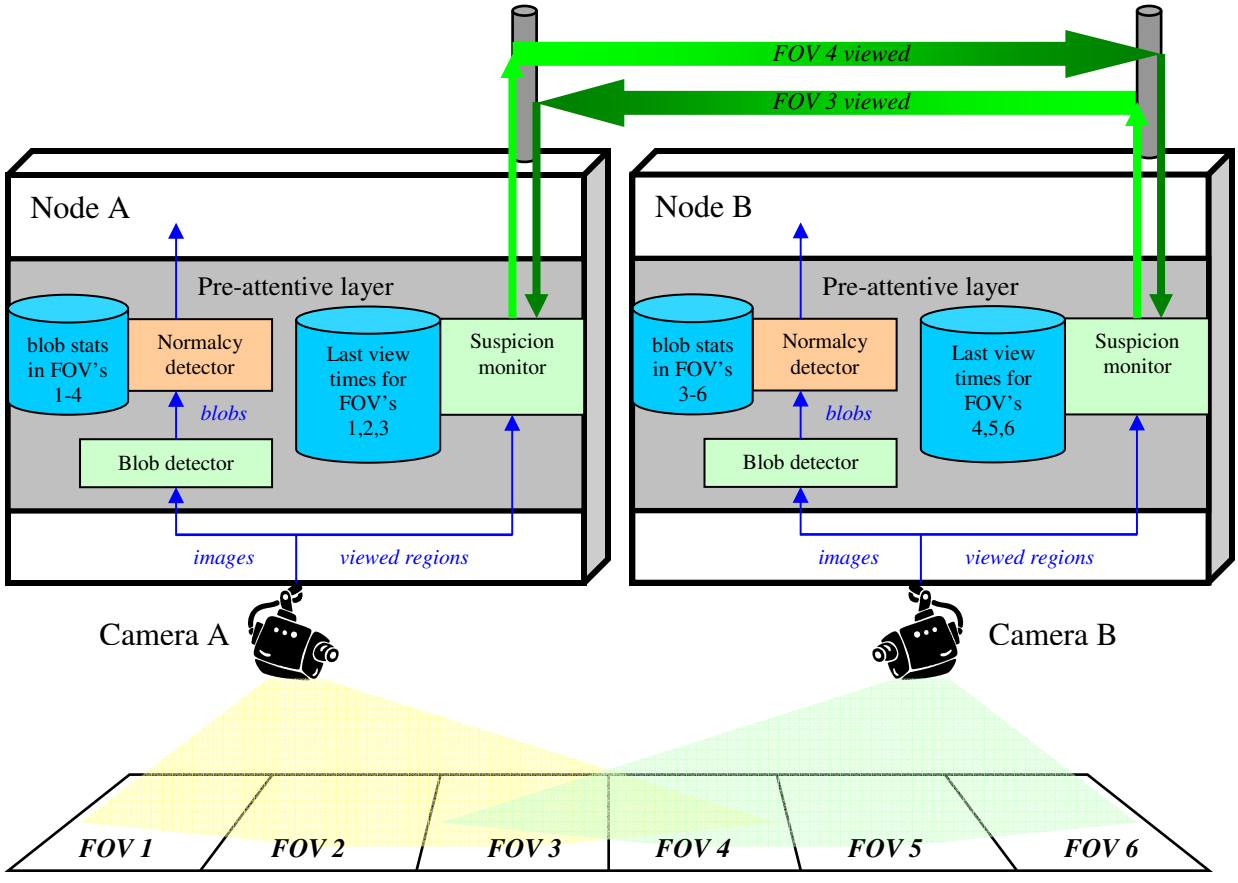


Figure 8. Distributed implementation of pre-attentive layer tasks

Distributed Implementation of the Pre-attentive Layer

An example of the distributed implementation of the blob detector, normalcy detector, and suspicion monitor is shown in Figure 8. The figure shows two nodes, labeled A and B, each with a camera, also labeled A and B. The world is a single lane of 6 FOV's numbered consecutively from left to right. Camera A can view FOV's 1 thru 4 while camera B can view FOV's 3 thru 6. Note that both nodes include all the sensemaking modules of the pre-attentive layer (See Figure 7). The blob detector and normalcy detector detect blobs and calculate normalcy from their local camera only. Note no communication between the blob detectors and normalcy detectors of nodes A and B are necessary because they do not need to share any information. The suspicion monitor module has been implemented in a distributed fashion by separating which FOV regions the suspicion monitors on nodes A and B are to handle. The suspicion monitor on node A is responsible for monitoring the suspicion in FOV's 1, 2, and 3 while the suspicion monitor on node B is responsible for monitoring the suspicion in FOV's 4, 5, and 6. Since the camera B can view FOV 3, then whenever the suspicion monitor on node B notices that camera B has viewed FOV 3, this information needs to be sent to the suspicion monitor on node A so that the suspicion about FOV 3 can be updated. This is shown graphically in Figure 8 by the green arrow coming out of the suspicion monitor on node B which forwards the event "FOV 3 viewed" to the suspicion monitor on node A. Analogously, the suspicion monitor on node A will forward the event that camera A has viewed FOV 4 to the suspicion monitor on node B. In our testbed, we have pre-wired which FOV's the suspicion monitor on each node is responsible. Future extensions could involve negotiations between nodes to set up this division of responsibility and to handle node failures.

Holistic View of the Attentive Layer

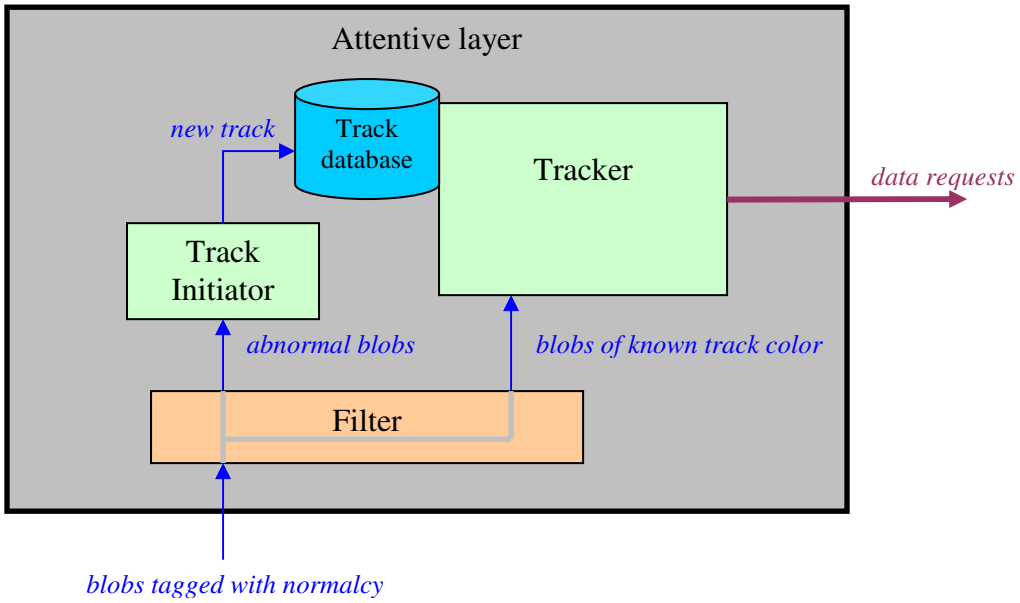
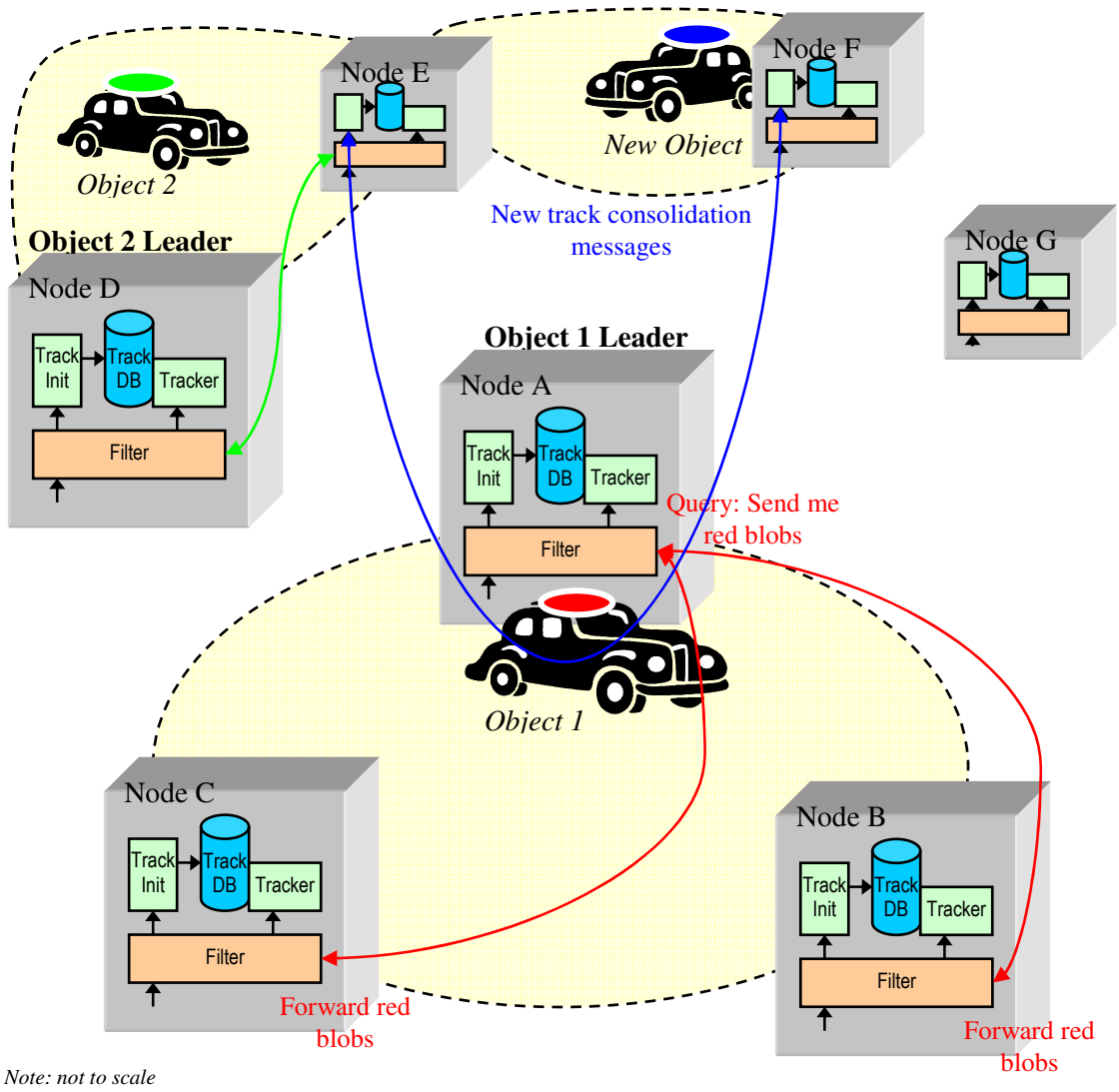


Figure 9. Attentive layer

The computations in the attentive layer create representations whose semantics are regarding the state of single, localized phenomena in the world, where the two key words are *single* and *localized*. Sensemaking tasks like tracking the motion of objects through the sensor network is an attentive layer task since object trajectories have a clear meaning regarding the motion of a *single*

phenomenon and the sensor data required for computation is *localized* in time and space. Figure 9 shows the attentive layer sensemaking modules in the camera testbed. The attentive layer consists of a tracker based on location estimates of colored blobs which have been computed by the pre-attentive layer. The color of the blob represents a discriminating feature which identifies a particular object, so that tracking multiple objects is possible in our system as long as the color of the objects are distinct. (footnote: Future work extending this testbed to a real application would involve developing a real discriminating feature set for identifying objects.) A new track is initiated by a track initiator module for blobs that are tagged as abnormal and have a color which is not the color of some previously tracked object. Thus, there is a filter module which determines whether to send blobs to the track initiator or to the tracker. Once a track is initiated and the color of the object identified, future blobs of the object's color will be incorporated into the track. Tracking is currently implemented with a particle filter.

An appropriate distributed implementation of tasks at the attentive layer is a leader-based approach. That is, a single node is elected the leader which is responsible for performing all the processing necessary to maintain the state of the object. This leader must also gather the appropriate data from sensors on remote nodes for processing. Such a quasi-centralized scheme may not seem scalable; however, this leader-based implementation is scalable because the sensors that have the most informative data regarding the object state tend to be spatially localized. Thus, by choosing the leader to be near (in terms of some notion of communication distance) the set of sensor nodes with good data, this leader-based implementation is scalable. For wireless communications, "nearness" in communication distance is generally correlated with "nearness" in spatial location. The set of sensors which have the most informative data can change over time. For example, as an object moves through a sensor network and assuming that the sensors which are near the object's position tend to have better data, then the set of sensors with good data changes over time. Thus, the leader node should change over time so that communicating the needed sensor data to the leader does not require communicating across an unnecessarily large part of the network. In our testbed, tracking the motion of objects is a good example which is amenable to this leader-based distributed implementation. By assigning a leader for each track, we can track multiple objects through the sensor network (as long as we do not have to deal with mixing identities). Note that a node can be the leader of multiple tracks. Cross node communications in the attentive layer are more dynamic involving changing data sinks (the leaders) and changing data sources (the spatially localized set of sensors) depending on the dynamics of an object's state evolution.



Note: not to scale

Figure 10. Distribute implementation of attentive layer tasks

Distributed Implementation of the Attentive Layer

In our camera testbed, the distributed implementation of the attentive layer is to replicate the modules in Figure 9onto every node of the sensor network as shown in Figure 10. To aid this discussion, when we refer to a sensemaking module of a particular node, we will refer to it according to the identifying node letter. For example, the filter module on node A will be referred to as filter module A, and the tracker module on node B will be referred to as tracker module B, or simply tracker B.

Let us first describe tracking and later describe track initiation. The track database of a node contains only the set of tracks for which this node is the leader. For example, node A is the leader node of object 1, so that track database A consists of the state representation of object 1. Node D is the leader of object 2, so that track database D consists of the state representation of object 2. All other nodes have no tracks in their track databases. The information flowing through the incoming arrow of the filter module of each node is the set of local blobs computed by the local node's blob detector in the pre-attentive layer. Since the set of nodes that have good data about object 1 are those nodes which are near it, as indicated by the dotted lined yellow ellipse around

the object, node A should get red blobs from nodes B and C whenever they appear. Tracker A accomplishes this by examining the current estimated position of object 1 and determining those sensor nodes which are close to the object. We assume that every sensor node already has the needed configuration information about neighboring sensor nodes so that tracker A requires no additional communications to determine this set of nodes. Then, tracker A will inform its local filter module A that it is interested in red blobs from nodes A, B, and C, since object 1 can be identified by a red blob. Filter module A will then send a message to filter modules B and C asking them for any red blobs that they encounter. When filter module A receives a red blob, it will send the red blob to tracker A for processing. Note that this prevents red abnormal blobs from initiating a new track on node A. When filter modules B and C receive a red blob, they will forward the red blob to filter module A, which will then send the red blob to tracker A. Note that this prevents red abnormal blobs from initiating a new track on nodes B and C as well. As data is collected, the best position estimate of object 1 may approach another node, say node B. At this point, tracker A will decide to make node B the leader of object 1. This requires, retracting interest of red blobs by filter module A and sending the state representation of object 1 to node B. When node B receives the track, it can then set up the appropriate queries to the set of nodes near object 1, and tracking of object 1 can continue in like manner as before. This leader-based implementation is on a per-track basis so that node D behaves analogously for object 2. Furthermore, there is no restriction that a sensor node can be the leader for any one track. The above implementation works for multiple tracks in the track database.

Now, let us describe track initiation. Say a blue target moving abnormally comes into range of sensor nodes E and F. Filter modules E and F may encounter a blue blob which is tagged abnormal by their respective normalcy detector in the pre-attentive layer. Since neither filter modules E and F have been notified of a blue object that has already been tracked in the world, they will send their local blue blobs to their local track initiators. If track initiators E and F both initiate a track for blue blobs, then we encounter a situation where there are duplicate tracks in the sensor network for the same object. Thus, track initiation requires a consolidation scheme so that the initiation of duplicate tracks can be prevented. Our solution is for track initiators E and F to send out a message indicating that they have noticed an abnormal blue blob. After a short time, the track initiators then know the set of nodes which have also observed an abnormal blue blob. Then, the track initiator which is on the sensor node that is closest to the set of abnormal blue blobs will be the one to actually initiate a new track. If there are any ties, then these can be broken via an ordering on the sensor node id, which is assumed to be unique. Note that this consolidation scheme requires that each sensor node know about its neighboring sensor nodes, which we assume was the case. This concludes the distributed implementation of the attentive layer tasks.

Holistic View of the Cognitive Layer

The computations in the cognitive layer create representations which can describe the behavior of groups of objects or phenomena which are not localized in the world. These sensemaking tasks potentially require information from sensor nodes which can be widely spaced apart. Thus, a leader-based distributed implementation may not be appropriate for tasks at this layer. What kinds of distributed implementations are suitable is a topic of future research which will require some example tasks to drive this exploration. The camera testbed currently does not implement any cognitive layer tasks.

Attention Mechanism in the Sensemaking Part

We will describe which pieces of the sensemaking part of the architecture involve implementing an attention mechanism. We will base this discussion on the holistic view of the system because the distributed view of the system includes extra issues which tend to confuse what parts of the architecture were designed to implement a focus of attention mechanism and what parts of the architecture was designed to handle the constraints of a distributed system. The attention mechanism involves two aspects on the sensemaking part. The first aspect is to limit what processing occurs.

The pre-attentive layer computes normalcy for all blobs that are detected by the blob detector. When these blobs are passed up to the attentive layer, they go through a filter which either (1) ignores them, (2) sends them to the track initiator, or (3) sends them to the tracker for updating a known track. Initially, the filter ignores all blobs that are not considered abnormal, and all blobs that are tagged to be abnormal are sent to the track initiator. After a new track has been created and sent to the tracker, the tracker will send a control message (along the red arrow in Figure 11) to the filter indicating what color blobs to send to the tracker in the future regardless of whether the blob is abnormal or not. In this way, future abnormal blobs of this track's color will not instantiate a duplicate track. Furthermore, if a track ever terminates, then the tracker can indicate to the filter that blobs of the terminated track's color should no longer be sent to the tracker. By controlling what data is sent where and since each module performs processing only when data is sent to it, the system is using processing resources only when needed. Note that the pre-attentive layer always processes images into blobs and runs the normalcy detector. There must be some computation which must always be executed to start the whole sensemaking process. Thus, it is desirable to make the computations at the lowest layers as inexpensive as possible, both in terms of processing and communication resource expenditure.

The second aspect is to suggest to the sensor system part what data should be collected by the sensors. The suspicion monitor at the pre-attentive layer sends data requests to view certain regions of the world with a priority that is proportional to the amount of time that has elapsed since the last camera observation. The tracker sends data requests to view each track that it is tracking. The priority for viewing a certain region depends on the importance of viewing the object and the probability of finding the object in the region.

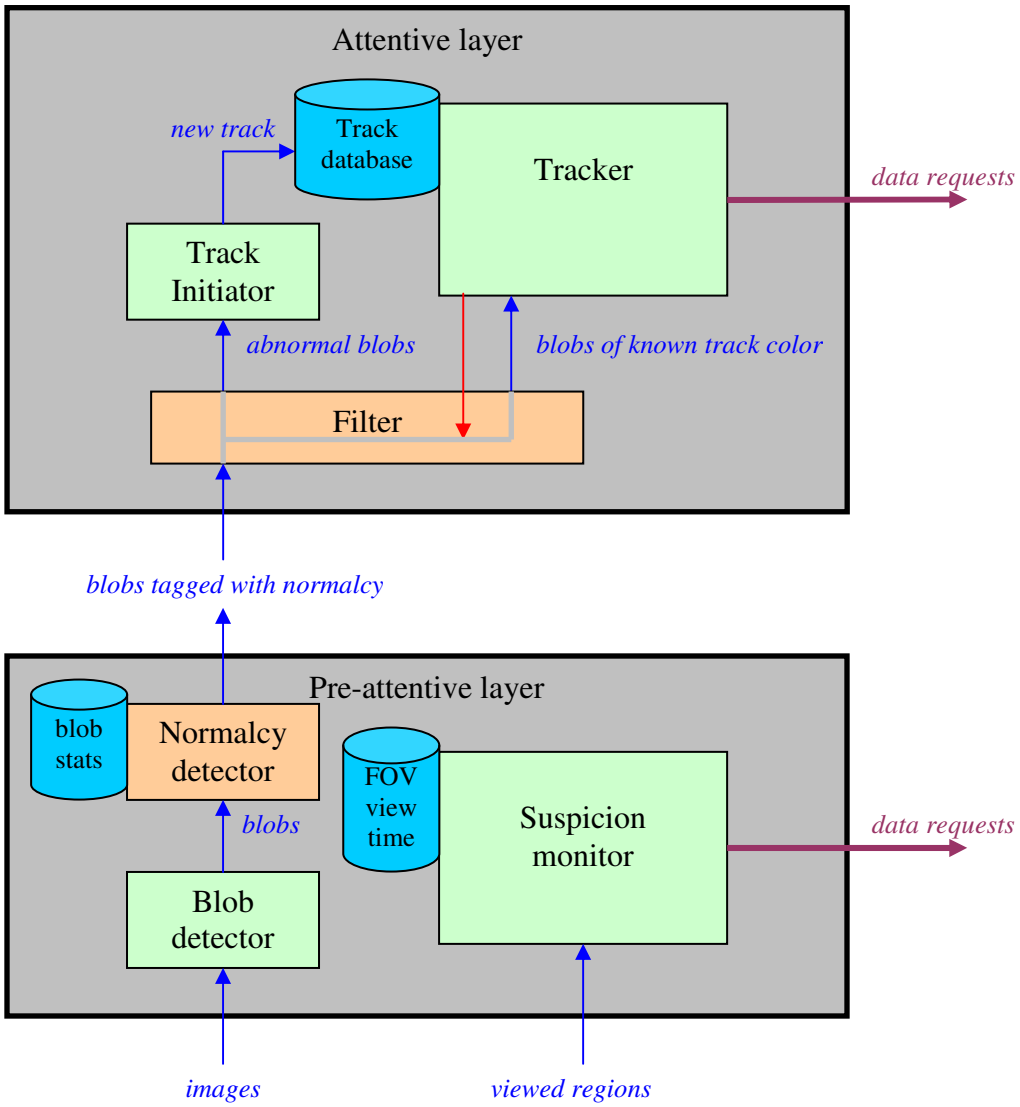


Figure 11. The pre-attentive and attentive layers of processing stacked up

1.2 Sensor System Part

The sensor system part is responsible for the following two functionalities:

1. resolving data requests and mapping requests to the appropriate sensors on the appropriate nodes and
2. maintaining an awareness of the set of sensor nodes.

The first functionality is embodied in the sensing resource allocation module of the system as shown in Figure 6. Our pan-tilt camera network has constraints on what set of regions can be viewed at any one time due to the limited number of regions each camera may view. Since it is possible that the data requests from the sensemaking part of the system may be a set of requests which cannot all be handled, the sensing resource allocation part of the system is needed to decide which requests to honor and which to ignore. In the camera testbed, the tracker requests for image data from those regions of the testbed where there is a high probability of viewing one of its tracks. Furthermore, the suspicion monitor requests for cameras to view certain regions with different priority. Thus, the sensing resource allocator chooses the best set of regions for the

cameras to view given the data requests from the sensemaking part of the system. The details of the resource allocation module are described in a subsection below.

In order to make a sensing resource allocation decision, not only must the set of data requests be provided but the set of available sensing resources must also be known. If the availability of sensor resources is dynamic, like if nodes can fail or new nodes can be added, then maintaining knowledge of the available resources is important. This second functionality is embodied in the resource awareness module shown in Figure 6. In the current camera testbed, when a sensor node is turned on, it will announce its arrival to the other nodes in the network. The nodes of the network will then exchange information about the set of regions that their cameras can view. The extent of this exchange of information can vary depending on the needs of the application, which has implications on the scalability of this approach.

Sensing resource allocation module

We will describe the sensing resource allocation module in detail in this subsection. In our particular camera testbed, the resource allocation module is responsible for pointing the pan-tilt cameras (PTC's) to view the set of FOV's which are considered the highest priority according to the data requests generated by the sensemaking tasks. The feasible set of FOV's which can be viewed is constrained by the limited set of FOV's each camera can view.

The suspicion monitor and tracker communicate their priorities for viewing particular FOV's in the testbed by assigning utility values to these data requests. The higher the utility value, the higher the priority for viewing the FOV. For example, the suspicion monitor will request to view an FOV with a utility equal to a constant multiplied by the elapsed time since any camera had last viewed the FOV. Thus, those FOV's which have not been viewed in a long time get larger utilities. To monitor a track and remembering that we have a particle representation of the location of the track, the utility assigned to view a particular FOV by this track is equal to a constant multiplied by the number of particles present in the FOV. Our particle filter implementation of the track state varies the number of particles used in proportion with the size of the uncertainty region of the track. Thus, targets with greater uncertainty tend to have higher priority than those with less uncertainty, which seems reasonable if we are to maintain an awareness of all known tracks. The total utility of viewing an FOV is defined to be the sum of the utilities assigned by each track and the suspicion monitor. Although we can combine the individual utilities in ways other than addition, we will see that addition will be amenable to a distributed solution using a variant of the sum-product algorithm on factor graphs.

Problem Statement

Our resource allocation problem for steering cameras is posed as a utility maximization problem. Let S be the set of all viewable FOV's in the world, and let $S_i \subset S$ be the subset of FOV's viewable by camera $i \in I$, where $I = \{1,\ldots,n\}$ is an index set of all n cameras in the system. Let $g : S \to \Re$ be the global utility value assigned to each FOV $s \in S$. Then, we wish to choose an FOV assignment $\hat{s}_i \in S_i$ for each camera $i \in I$, so that we maximize

$$h(\hat{s}_1,\ldots,\hat{s}_n) = \sum_{a \in \hat{S}} g(a)$$

where $\hat{S} = \bigcup_{i \in I} \{\hat{s}_i\}$. Note that if multiple cameras are assigned to view the same FOV, then the utility value for that FOV is *not* counted multiple times.

Since our system is distributed, we need a distributed method for solving, at least near optimally, the maximization problem above. Although we could have posed this problem as an auction, we choose an approach based on *factor graphs*, which can be generalized to handling combinatorial auctions. We use the max-sum algorithm, a variant of the sum-product algorithm, to approximately compute the maximizing assignment of cameras to FOVs. This transformation allows us to derive a distributed algorithm for resource allocation.

We wish to rewrite the function h above explicitly as a sum of the individual utility values assigned by each inference task. To facilitate explanation, we will use the example of Figure 12. The example is a world consisting of four FOV's with three cameras, where camera 1 can view either FOV 1 or FOV 2, camera 2 can view either FOV 2 or FOV 3, and camera 3 can view either FOV 3 or FOV 4. Let $J = \{1,\ldots,4\}$ be the index set of individual inference tasks assigning utility values to the FOV's, which in this example are: (1) the suspicion monitor for FOV 1 which currently shows a suspicion level of 2.0, (2) the suspicion monitor for FOV 2 showing 1.0, (3) the suspicion monitor for FOV 3 showing 1.5, (4) the suspicion monitor for FOV 4 showing 4.0, and (5) the track for target 1 with gray particles representing possible locations of the target.

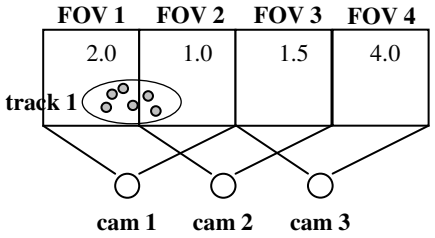


Figure 12. Resource allocation example

Each of these inference tasks assigns utility values for the FOV's and hence, utility values for certain assignments of cameras to FOV's. Let $s_1 \in \{1,2\}$ be the FOV (either FOV 1 or FOV 2) that camera 1 can view, $s_2 \in \{2,3\}$ be the FOV (either FOV 2 or FOV 3) that camera 2 can view, and $s_3 \in \{3,4\}$ be the FOV (either FOV 3 or FOV 4) that camera 3 can view. The following functions $\{f_i\}_{i=1}^5$ are the utility values contributed by each individual inference task i depending on the FOV assigned to the cameras.

1. For the suspicion monitor task of FOV 1, since only camera 1 can view FOV 1, f_1 is a function of only s_1. Letting the utility for viewing FOV 1 be the current suspicion level, we have

$$f_1(s_1) = \begin{cases} 2.0 & s_1 = 1 \\ 0.0 & s_1 = 2 \end{cases}.$$

2. For the suspicion monitor task of FOV 2, since FOV 2 can be viewed by both camera 1 and camera 2, the utility function f_2 is dependent on both s_1 and s_2. We have

$$f_2(s_1, s_2) = \begin{cases} 1.0 & s_1 = 1, s_2 = 2 \\ 0.0 & s_1 = 1, s_2 = 3 \\ 1.0 & s_1 = 2, s_2 = 2 \\ 1.0 & s_1 = 2, s_2 = 3 \end{cases}$$

for the utility contributed by this task. Note that we do not double-count the utility if both camera 1 and camera 2 view FOV 2.

3. For the suspicion monitor task of FOV 3, since FOV 3 can be viewed by both camera 2 and camera 3, we have

$$f_3(s_2, s_3) = \begin{cases} 1.5 & s_2 = 2, s_3 = 3 \\ 0.0 & s_2 = 2, s_3 = 4 \\ 1.5 & s_2 = 3, s_3 = 3 \\ 1.5 & s_2 = 3, s_3 = 4 \end{cases}.$$

4. For the suspicion monitor task of FOV 4, since only camera 3 can view FOV 4, we have

$$f_4(s_3) = \begin{cases} 0.0 & s_3 = 3 \\ 4.0 & s_3 = 4 \end{cases}.$$

5. For the tracking task, we first look for the set of cameras that can view any particle of the track so that the function f_4 is a function of both s_1 and s_2. For this example, assigning a utility value of 1 to every particle that is viewed gives us

$$f_5(s_1, s_2) = \begin{cases} 6.0 & s_1 = 1, s_2 = 2 \\ 4.0 & s_1 = 1, s_2 = 3 \\ 2.0 & s_1 = 2, s_2 = 2 \\ 2.0 & s_1 = 2, s_2 = 3 \end{cases}.$$

Thus, we have rewritten the global utility function h as

$$f_1(s_1) + f_2(s_1, s_2) + f_3(s_2, s_3) + f_4(s_3) + f_5(s_1, s_2)$$.

Note that h is a function of three variables, but it can be written as the sum of functions of less than three variables each. It is this special structure that factor graphs can take advantage of for efficient computation. For our particular resource allocation problem, we are interested in computing

$$\arg_{s_1, s_2, s_3} \max f_1(s_1) + f_2(s_1, s_2) + f_3(s_2, s_3) + f_4(s_3) + f_5(s_1, s_2)$$.

The graphical picture of this "factorization" of h is shown in Figure 13.

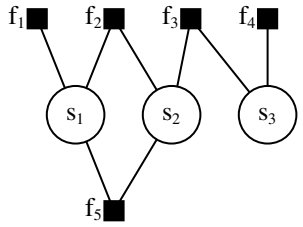


Figure 13. The factor graph

A factor graph is a bipartite graph with two kinds of nodes, function nodes as denoted by black squares and variable nodes as denoted by the white circles. Each function node is connected to

the set of variables that it depends, capturing the "factorized" form of the overall function h that it represents. The max-sum algorithm is an iterative message passing algorithm where messages are passed along the edges of the factor graph and summary operations are computed at the function and variable nodes of the graph. For factor graphs that contain no cycles, the result of these iterations is the maximizing variable assignment. For factor graphs with cycles, there are few guarantees about the result that are known theoretically , although in practice, the results have been promising.

For our distributed attention system, it is important that we have a distributed algorithm for performing this resource allocation. Recall that each inference task generates a utility function f_i which is a function of the FOV assignments of all cameras which can provide data for this inference task. Our idea is for each inference task to communicate the utility function f_i to a local set of camera nodes. Then, each camera can execute the max-sum algorithm and point itself to what it believes is the best direction. To be globally consistent, each inference task would have to communicate its utility function to all cameras in the entire network. However, since it is only those cameras which are near each other that really must negotiate and less so when a camera is very far away, the utility function for each inference task need only be communicated to a local neighborhood of cameras in practice.

2 System Architecture Design Considerations

This section describes the implications of drawing the boundary between the sensemaking part and the sensor system part as shown in Figure 6.

One reason is to separate the expertise needed to develop such a multi-disciplinary system. The sensemaking part involves expertise in signal processing, detection and estimation theory, recursive estimation, and high-level AI. On a sensor network, this part is the application part of the system, and the information processing algorithms are meant to be application specific. The sensor system part involves expertise in distributed systems, embedded systems, and real-time scheduling and allocation. This part is meant to be application generic and provides an abstraction of the machinery below on top of which an information processing application can be developed.

Another reason is the design modularity that this boundary enables. By keeping resource allocation and resource awareness separate from the sensemaking algorithms, the sensemaking algorithm implementations do not need to be modified if sensor nodes fail or new sensor nodes are added. A change in the particular networking infrastructure, either wireless or wired, does not require a modification of the sensemaking code. This boundary makes the sensemaking algorithms generic over sensor network hardware configurations. However, this modularity comes at the cost of disallowing sensemaking algorithms to have direct control over how particular sensing devices are used. Such capabilities could be useful for sensemaking algorithms which are resource aware, referring to those algorithms that adapt to the available resources. We believe a middle ground between allowing sensemaking algorithms to have complete access to the sensing devices versus enforcing a strict boundary, where sensemaking algorithms have no control of the sensing devices used, is to develop a more detailed data request description so that the sensemaking algorithms can have some control about their preferences in sensing devices used coupled with a feedback loop from the sensing resource allocator which provides high-level descriptions of the available sensing devices, which could be utilized by resource aware sensemaking algorithms.

Appendix II. Performance metrics

We identified a few metrics for performance evaluation of distributed attention systems. The metrics can be roughly categorized into two classes: one for evaluating sensing and tracking qualities of the attentional system, and the other for evaluating the quality of distributed system engineering, such as reliability, resource allocation efficiency etc.

Sensing and tracking metrics

The sensing and tracking metrics we identified includes:

- Detection
  - ***Time-to-detection*:** it measures the time it takes from when the ground truth of when an anomaly first appears in the world to the time it takes for the system to sense it. This metric isolates the particular choice of peripheral awareness monitoring from the detection algorithm.
  - ***False alarm rate and missed detection rate:*** these are standard performance metrics tooted in estimation theory. There is a tradeoff between the two metrics, often viewed in a ROC (Receiver Operating Characteristic) curve, which is a graph plotting the detection probability versus the probability of false alarm. See [Shanmugan-1988, pg.352] for a reference.
- Classification
  - ***Misclassification rate***: it measures the accuracy of the classification method used with respect to specific application scenarios.
- Tracking
  - ***Track coverage*:** this measures the percentage of time that a track is covered by any camera. To be objective, the value of this metric should be conditioned on available resources, such as camera density.
  - ***Track loss rate***: this measures the percentage of established tracks that gets lost.
  - ***Track accuracy***: this measures the difference between the actual trace and the computed trace of a track.

Based on these metrics, inference algorithms can be optimized to achieve good performance. For example, various estimation algorithms, e.g., Neuman-Pearson detection, Bayesian estimation, and minimax estimation, have been proposed to match different optimality criteria. For most practical inference problems, designing a suitable inference algorithm is non-trivial and an active field of research.

System engineering metrics

In distributed attention systems, there is the additional tradeoff between multiple concurrent inference tasks due to their competition for resources. For example, Figure 14 illustrates the tradeoff within a system with a detection task and a tracking task. As the resource allocation scheme grants tracking more resources (moving down the curve), tracking quality (e.g., average variance) is improved, but the detection quality (false alarm or missed detection rate) deteriorates. A good resource allocation scheme should try to push the curve, as indicated by the arrows in Figure 14, in order to achieve an optimal balance between the tasks.

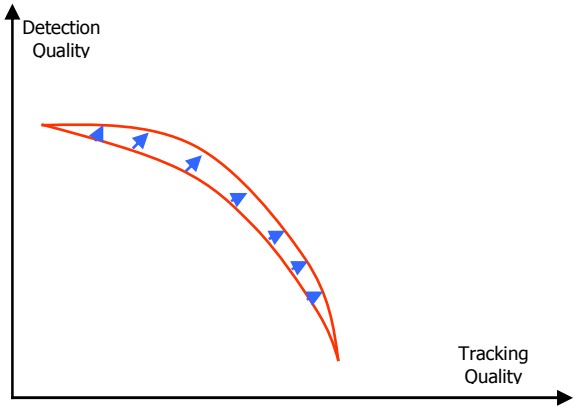


Figure 14. Tradeoff between detection and tracking with contention for fixed resources

Furthermore, under resource contention, instead of using performance metrics for each individual task, for convenience one may use summarized metrics based on application requirements. For example,

- To test the robustness of tracking against clutters, the performance metric can be the number of distractors that the system can tolerate while maintaining tracking quality above a given level. As shown in Figure 15, given fixed resource, a good engineering design should try to push the Quality-vs-distractor curve upward.

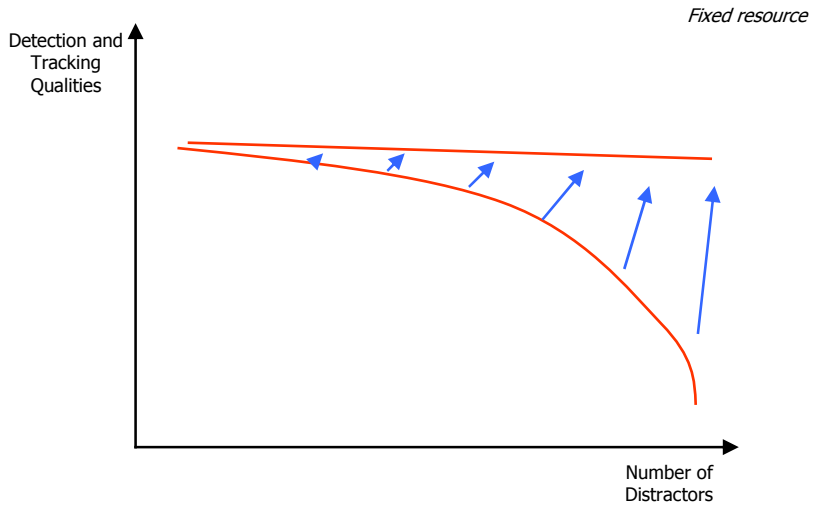


Figure 15. Engineering against the number of distractors

- In a system with real-time requirements, instead of using detection quality such as the false alarm or missed detection rate, one may use time-to-detection, which measures the amount of time that an abnormal target remains undetected since its first appearance. It turns out that time-to-detection distribution is a summarization of the detailed detection metrics. Similarly, we can measure time-to-classification, or generally, time that it takes to fulfill a task or verify/disprove a hypotheses.

To evaluate a resource allocation scheme, we can compare it against a few benchmarks:

- Round robin allocation, where sensing resources are granted to tasks in turn.
- Random allocation: sensing resources are allocated randomly according to some distribution. This scheme may in fact be optimal in special cases. For example, if the

abnormalities pop up at random locations at random times, and there is no correlation, random allocation is the best thing to do.

- Optimal allocation: This is the computationally expensive centralized allocation "oracle" scheme. Comparison with this optimal allocation shows how much performance is sacrificed by restraining the system to local communication.

Finally, the robustness of the system against component failures could be measured by the quantitative changes of the sensing and tracking metrics of the system when some cameras/sensors are shutdown. A good system should exhibit graceful degradation of the sensing and tracking qualities.

Appendix III. Experiments

In this appendix, we show experimental simulation results which demonstrate that our distributed attention system can be tuned to trade off detection of new abnormal behavior against persistent tracking of abnormal targets. We also compare our system to a baseline system where cameras are steered randomly.

Description

To measure how well emergent abnormal behavior is detected, we consider the *detection latency*, the interval of time (in units of sensing cycles) between the emergence of an abnormal behavior and its detection. To measure the track quality, we consider *track coverage*, the average percentage of the time a track is viewed by any camera. An ideal system is one which both minimizes detection latency and maximizes track coverage.

In our experiment, abnormal behaviors emerge according to a Poisson arrival process, and are then randomly positioned on any viewable part of the testbed. The detection latency is independent of the arrival rate since processing abnormal behavior is modeled as having negligible cost. We also have 0-6 targets moving at various speeds around the testbed in either a clockwise or counterclockwise direction. These targets are hard-wired to be abnormal from the start, while instantiation of new tracks is deactivated to allow us to measure the detection latency for various numbers of tracks. We then collect the detection latency and the track coverage as we vary the *relative track utility*, the ratio of track utility to suspicion utility and the number of tracked targets.

Detection latency CDF

Figure 16 and Figure 17 show the empirical detection latency cumulative distribution function (CDF) for the random camera allocation (dotted line), along with the CDFs (solid lines) for the utility-based resource allocation scheme described in Appendix I for up to 6 targets. As expected, when there are more targets being tracked by the system, the detection latency CDF shifts downward and to the right, meaning detection latency is getting longer. When the relative track utility is low as in Figure 16, added tracking tasks have less effect on the detection latency than when the relative track utility is high, as in Figure 17. When the relative tracking utility is high and a large number of vehicles are being tracked, the detection latency can be much worse than the randomly allocated case; however, this cost is traded off with better track coverage as we shall explain next.

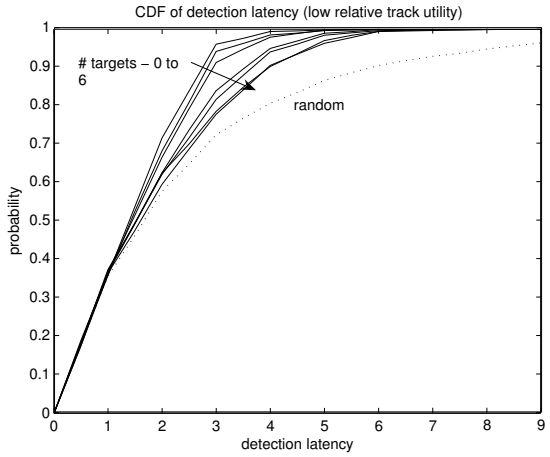


Figure 16. Detection latency (low track utility)

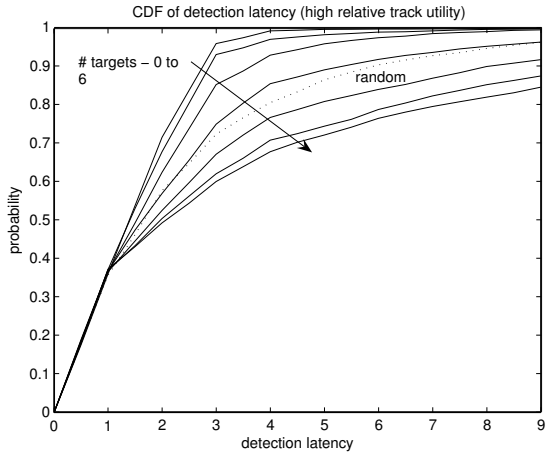


Figure 17. Detection latency (high track utility)

Tradeoff between mean detection time and track coverage

In Figure 18, we vary the relative track utility to show the tradeoff between track coverage and detection latency for various numbers of targets. When the relative track utility is small, then regardless of the number of tracks, the system has a mean detection time of around 1.5 cycles and track coverage slightly above 40%. As we increase the relative track utility, the track coverage increases at the cost of a longer mean detection time. As more tracks are added to the system, camera resources become scarcer, increasing the impact on detection time. Note that with one or two tracking tasks in the system, camera resources are sufficiently available that the impact on the mean detection latency is negligible. For comparison, the randomly allocated system, denoted by the stars, has a mean detection time of around 2.5 cycles and track coverage around 35% regardless of the number of tracks. This demonstrates the efficiency of the resource allocation scheme in balancing competing needs of the system.

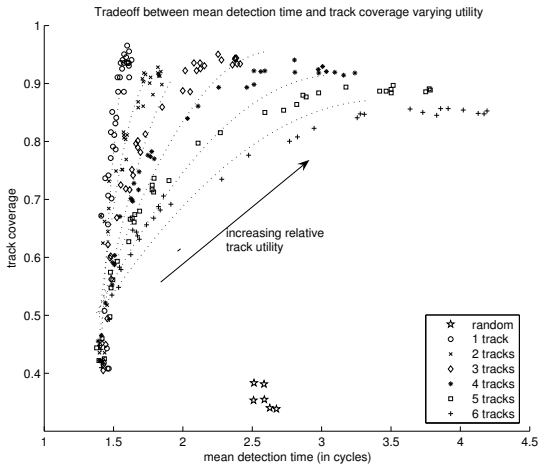


Figure 18. Trading off detection time vs. coverage